Intel® DPC++ Compatibility Tool

Migration Experiences Sharing

Lin Jie (Auber)

Plyakhin Yury

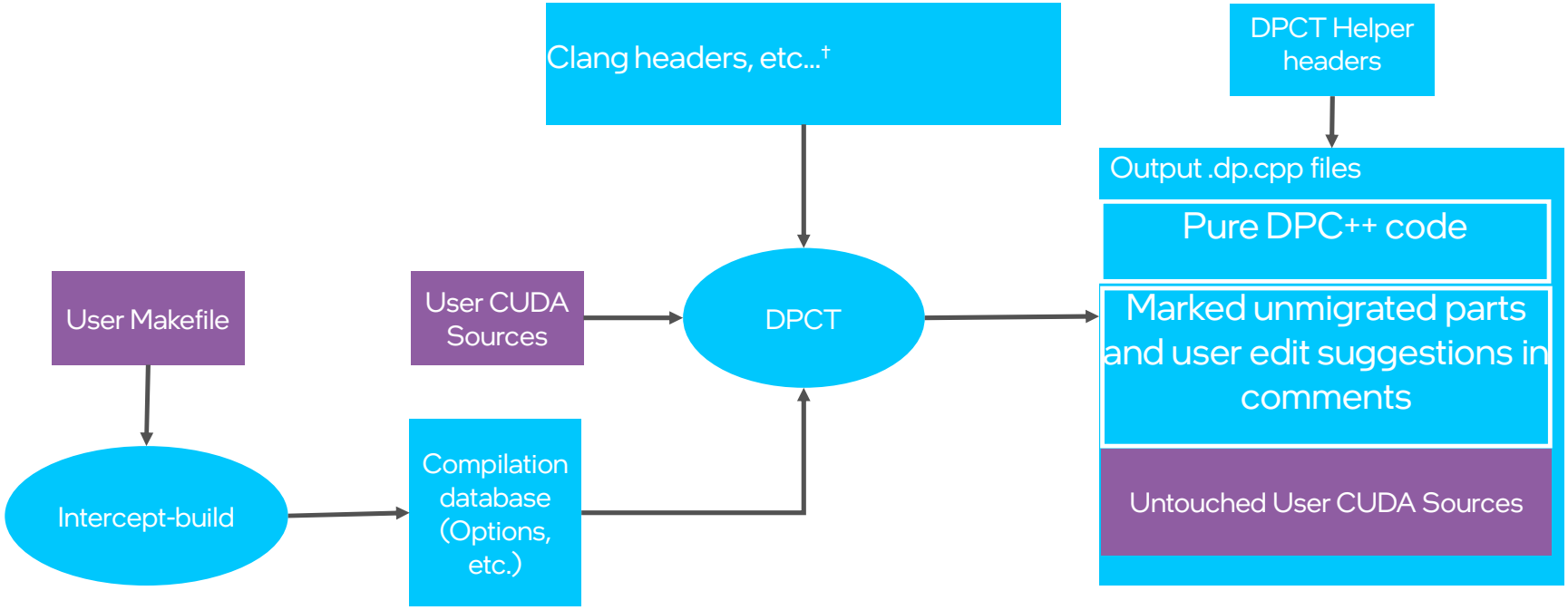
Jan.2022

Agenda

- High-level workflow
- Common practices in large project migration
- New features to ease build & release
- Texture/image code migration
- Experimental features
- Workarounds in original CUDA project

High-level workflow

High-level Workflow



Command line example:

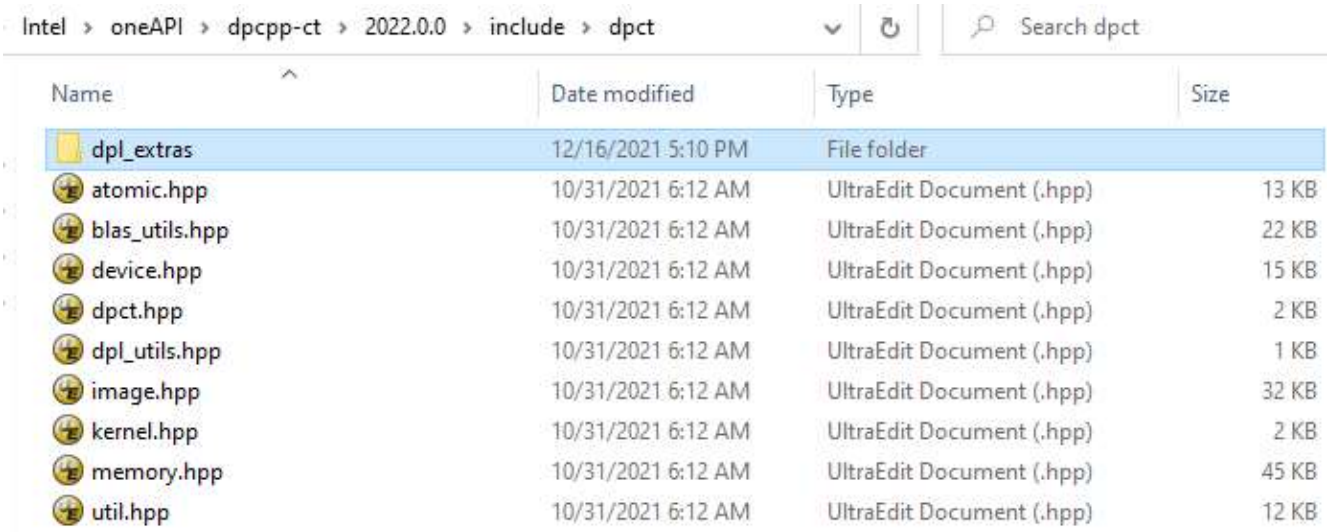
```
intercept-build make
```

Command line example:

```
cd rodinia_3.1/cuda/nw
dpct -p . --in-root=./ --out-root=dpct_output  --keep-original-code needle.cu
```

† Certain CUDA header files may need to be available

DPCT Helpers



- DPCT implements common code patterns to assist the migration of input source code as a set of header files.
- Uses SYCL style and oneAPI libraries, e.g., oneMKL, oneDPL

Migration of large cmake-style project

Basic Options of Intel® DPC++ Compatibility Tool

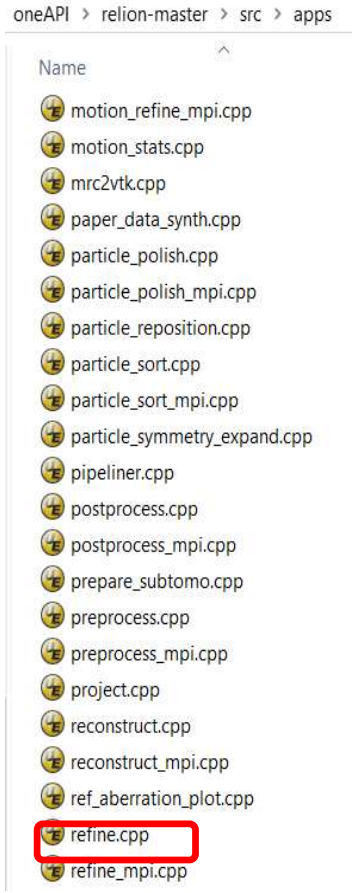
- --in-root=<dir>: path for the root of the source tree that needs to be migrated
- --out-root=<dir>: path for root of generated files (NOT specify the same with –in-root)
- -p=<dir>: path for the compilation database
- --process-all : Migrates or copies all files from the --in-root directory to the --out-root directory
- --keep-original-code : Keeps the original code in comments of generated DPC++ files
- --extra-arg=<string> : Specify more compiler options (The options that can be passed this way can be found with the *dpct -- -help* command.)

(Using *dpct --help* to get all the options of dpct)

Common Practices

- Know the build configuration components of the large CUDA project
  - Some configuration options are necessary to enable CUDA code base
- Get the compilation database via 'intercept-build make target'
  - 'target' limits the set of CUDA files to those you are interested in.
- Run dpct pointing to compilation database and '–process-all'
- Use dpcpp to compile output DPC++ code
- Integrate new DPC++ code base back to the large project

Relion migration



Configure and running with standard benchmarks 3D classification from https://www3.mrc-lmb.cam.ac.uk/relion/index.php/Benchmarks_%26_computer_hardware

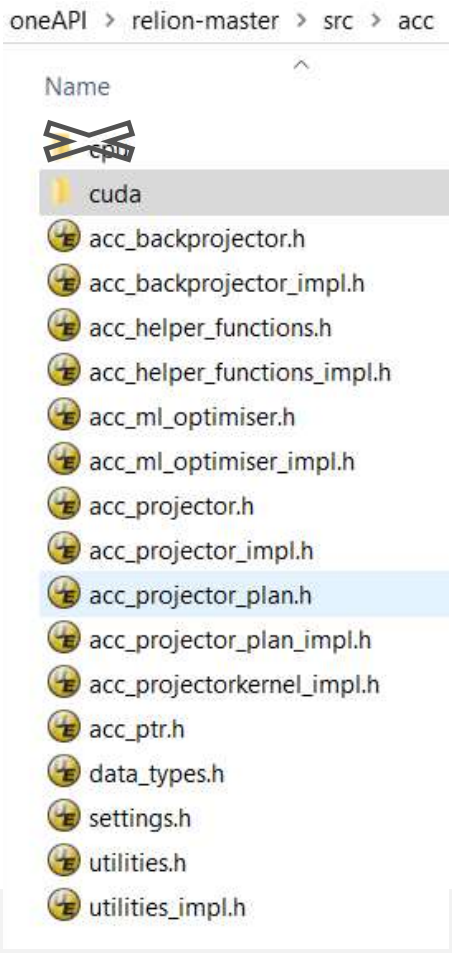
'CC=icx CXX=dpcpp cmake -DCUDA_ARCH=70 -DCUDA=ON -DCudaTexture=OFF -DGUI=OFF -DCMAKE_BUILD_TYPE=DEBUG -DBUILD_SHARED_LIBS=OFF -D CMAKE_C_FLAGS=" -O0 -g " -D CMAKE_CXX_FLAGS=" -O0 -g " ..'

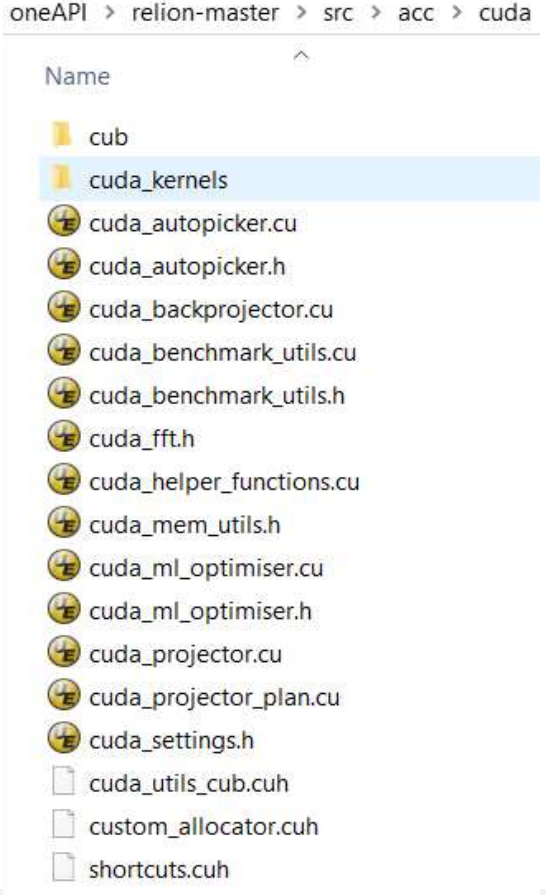
```
bin/relion_refine: src/apps/CMakeFiles/refine.dir/refine.cpp.o
bin/relion_refine: src/apps/CMakeFiles/refine.dir/build.make
bin/relion_refine: lib/librelion_lib.a
bin/relion_refine: /usr/local/cuda-10.1/lib64/libcufft.so
bin/relion_refine: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi_cxx.so
bin/relion_refine: /usr/lib/x86_64-linux-gnu/openmpi/lib/libmpi.so
bin/relion_refine: lib/librelion_gpu_util.a
bin/relion_refine: /usr/lib/x86_64-linux-gnu/libtiff.so
bin/relion_refine: /usr/lib/x86_64-linux-gnu/libtiff.so
bin/relion_refine: lib/librelion_lib.a
bin/relion_refine: lib/librelion_gpu_util.a
bin/relion_refine: /usr/lib/x86_64-linux-gnu/libfftw3f.so
bin/relion_refine: /usr/lib/x86_64-linux-gnu/libfftw3.so
bin/relion_refine: /usr/local/cuda-10.1/lib64/libcurand.so
bin/relion_refine: /usr/local/cuda-10.1/lib64/libcudart_static.a
bin/relion_refine: /usr/lib/x86_64-linux-gnu/librt.so
bin/relion_refine: /usr/local/cuda-10.1/lib64/libcufft.so
bin/relion_refine: src/apps/CMakeFiles/refine.dir/link.txt
```

build\src\apps\CMakeFiles\refine.dir\build.make

LIBRELION_GPU_UTIL

- agent_histogram.cuh
- agent_radix_sort_downsweep.cuh
- agent_radix_sort_upsweep.cuh
- agent_reduce.cuh
- agent_reduce_by_key.cuh
- agent_rle.cuh
- agent_scan.cuh
- agent_segment_fixup.cuh
- agent_select_if.cuh
- agent_spmv_csrt.cuh
- agent_spmv_orig.cuh
- agent_spmv_row_based.cuh
- arg_index_input_iterator.cuh
- block_adjacent_difference.cuh
- block_discontinuity.cuh
- block_exchange.cuh
- block_histogram.cuh
- block_histogram_atomic.cuh
- block_histogram_sort.cuh
- block_load.cuh
- block_radix_rank.cuh
- block_radix_sort.cuh
- block_raking_layout.cuh
- block_reduce.cuh
- block_reduce_raking.cuh
- block_reduce_raking_commutative_only.cuh
- block_reduce_warp_reductions.cuh
- block_scan.cuh
- block_scan_raking.cuh
- block_scan_warp_scans.cuh
- block_scan_warp_scans2.cuh
- block_scan_warp_scans3.cuh
- block_shuffle.cuh
- block_store.cuh
- BP.cuh
- cache_modified_input_iterator.cuh
- cache_modified_output_iterator.cuh
- constant_input_iterator.cuh
- counting_input_iterator.cuh
- cub.cuh
- cuda_device_utils.cuh
- cuda_utils_cub.cuh
- custom_allocator.cuh
- device_histogram.cuh
- device_partition.cuh
- device_radix_sort.cuh
- device_reduce.cuh
- device_run_length_encode.cuh
- device_scan.cuh
- device_segmented_radix_sort.cuh
- device_segmented_reduce.cuh
- device_select.cuh
- device_spmv.cuh
- diff2.cuh
- discard_output_iterator.cuh
- dispatch_histogram.cuh
- dispatch_radix_sort.cuh
- dispatch_reduce.cuh
- dispatch_reduce_by_key.cuh
- dispatch_rle.cuh
- dispatch_scan.cuh
- dispatch_select_if.cuh
- dispatch_spmv_csrt.cuh
- dispatch_spmv_orig.cuh
- dispatch_spmv_row_based.cuh
- grid_barrier.cuh
- grid_even_share.cuh
- grid_mapping.cuh
- grid_queue.cuh
- helper.cuh
- mutex.cuh
- shortcuts.cuh
- single_pass_scan_operators.cuh
- tex_obj_input_iterator.cuh
- tex_ref_input_iterator.cuh
- thread_load.cuh
- thread_operators.cuh
- thread_reduce.cuh
- thread_scan.cuh
- thread_search.cuh
- thread_store.cuh
- transform_input_iterator.cuh
- util_allocator.cuh
- util_arch.cuh
- util_debug.cuh
- util_device.cuh
- util_macro.cuh
- util_namespace.cuh
- util_ptx.cuh
- util_type.cuh
- warp_reduce.cuh
- warp_reduce_shfl.cuh
- warp_reduce_smem.cuh
- warp_scan.cuh
- warp_scan_shfl.cuh
- warp_scan_smem.cuh
- wavg.cuh





Migrate and Build

- make clean
- intercept-build make relion_gpu_util
- dpct -p=./build --in-root=./ --out-root=dpct_output --keep-original-code –process-all
- Review/update manually the resulting DPC++ code.
  - Rename MACROs from original CUDA code e.g. replace 'CUDA' with 'DPCPP'
  - DPCT automatically replaces the '__CUDA_ARCH__' with 'DPCT_COMPATIBILITY_TEMP', but no more.
- dpcpp -g -O0 -c *.dp.cpp cuda_kernels/helper.dp.cpp -I[root of relion] -I[root of relion]/external/fftw/include –DDPCPP(was -DCUDA) -DPROJECTOR_NO_TEXTURES -DHAVE_SINCOS -DHAVE_TIFF
- ar cr librelion_gpu_util.a *.o
- make refine（host part build）
- dpcpp -g -O0 -DDEBUG_CUDA -fiopenmp -rdynamic ./src/apps/CMakeFiles/refine.dir/refine.cpp.o -o ./bin/relion_refine -Wl,-rpath,[root of relion]/external/fftw/lib: ./lib/librelion_lib.a -ltiff ./lib/librelion_lib.a ./lib/librelion_gpu_util.a ../external/fftw/lib/libfftw3.so ../external/fftw/lib/libfftw3f.so -lpthread -ldl -lrt

Enhancements in build and release

New Options of Intel® DPC++ Compatibility Tool 2022.0

- --gen-build-script

  generates makefile for migrated file(s) in -out-root directory

- --build-script-file=<file>

  name of generated makefile for migrated file(s). Default name: Makefile.dpct.

- --use-custom-helper=<value>

  customize the DPCT helper header files for migrated code and place them in the --out-root directory. The values are: none, file, api, all

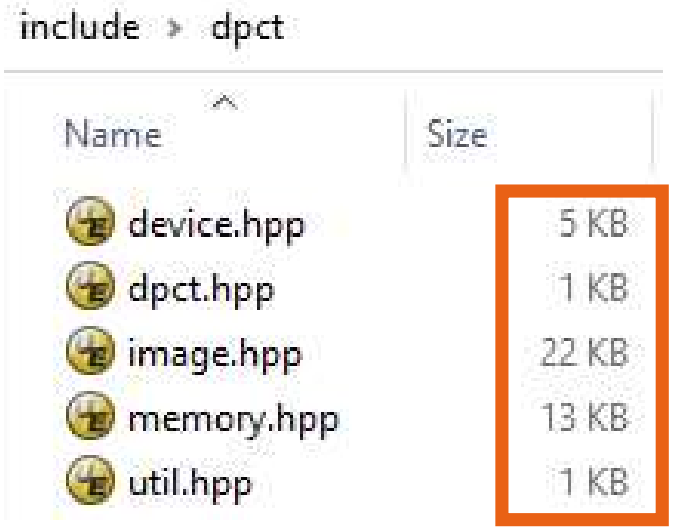
MAGMA project Makefile.dpct snippet via –gen-build-script

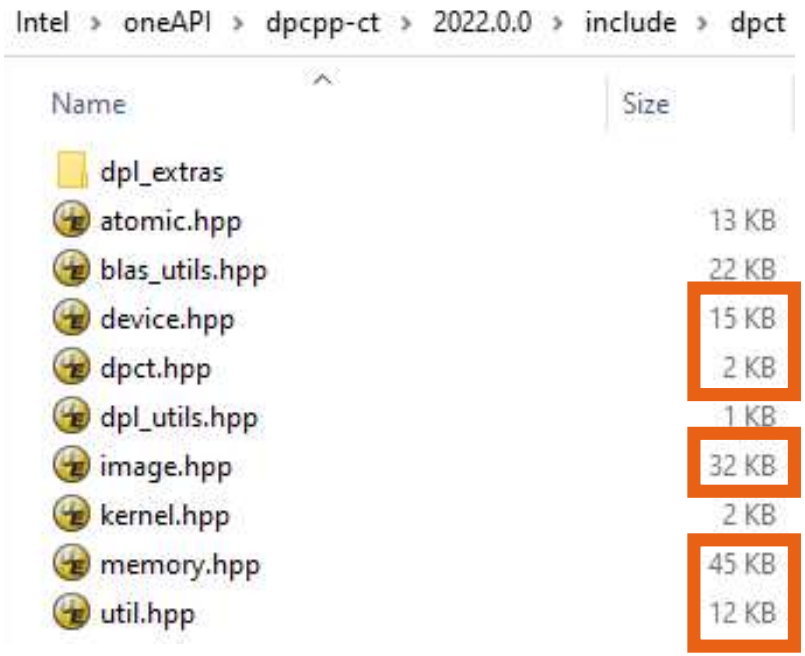
```
CC := dpcpp

LD := $(CC)

#DPCT2001:8814: You can link with more library by add them here.
LIB :=

FLAGS :=

TARGET_0_SRC_0 = ./control/abs.cpp.dp.cpp
TARGET_0_OBJ_0 = ./control/abs.cpp.dp.o
TARGET_0_FLAG_0 = -O3 -DADD_ -I./include -I./control  -I ./include ${FLAGS}
```

```
.PHONY:all clean
OBJS_0 :=  ${TARGET_0_OBJ_0} ${TARGET_0_OBJ_1} ${TARGET_0_OBJ_2} ${TARGET_0_OBJ_3} ${TARGET_0_OBJ_4} ${TARGET_0_O
${TARGET_0_OBJ_6} ${TARGET_0_OBJ_7} ${TARGET_0_OBJ_8} ${TARGET_0_OBJ_9} ${TARGET_0_OBJ_10} ${TARGET_0_OBJ_11}
${TARGET_0_OBJ_12} ${TARGET_0_OBJ_13} ${TARGET_0_OBJ_14} ${TARGET_0_OBJ_15} ${TARGET_0_OBJ_16} ${TARGET_0_OBJ_17}
${TARGET_0_OBJ_18} ${TARGET_0_OBJ_19} ${TARGET_0_OBJ_20} ${TARGET_0_OBJ_21} ${TARGET_0_OBJ_22} ${TARGET_0_OBJ_23}

OBJS :=  ${OBJS_0}

all: $(OBJS)

$(TARGET_0_OBJ_0):$(TARGET_0_SRC_0)
    $(CC) -c ${TARGET_0_SRC_0} -o ${TARGET_0_OBJ_0} $(TARGET_0_FLAG_0)
```

Release with only necessary DPCT helpers

```
dpct --out-root=dpct_output CopyTex2D.cu --use-custom-helper=api
```





Texture sample migration

SYCL image support limitation

- DPC++ RT supports 4-channel image format ONLY
- DPCT supports migration of more scenarios, than what is supported currently by SYCL
- Manual edit needed in DPCT output for both –
  - Image format
  - Input data layout

2D Texture in CUDA

```
texture<int, 2> textureD;
__global__ void kernel(int *dOutput,
int width)
{
  int row = threadIdx.y;
  int col = threadIdx.x;
  dOutput[row * width + col] =
tex2D(textureD, col, row);
}

int *d; size_t pitch;
cudaMallocPitch(&d, &pitch, …);
cudaChannelFormatDesc channel =
cudaCreateChannelDesc<int>();
cudaBindTexture2D(NULL, &textureD, d,
&channel, width, height, pitch);

kernel<<<1, (width, height)>>>(dOutput,
width);
```

Output from DPCT :

```
/*
DPCT1059:2: SYCL only supports 4-channel image format. Adjust the code.
*/
dpct::image_wrapper<int, 2> textureD;
/*
DPCT1059:3: SYCL only supports 4-channel image format. Adjust the code.
    */
dpct::image_channel channel =
dpct::image_channel::create<int>();
```

DPCT1059

DPCT1059

Message

SYCL only supports 4-channel image format. Adjust the code.

Detailed Help

SYCL* supports only 4-channel image format. The warning is emitted, when the tool generates code with unsupported image format, which corresponds to the original code. You can fix the resulting code by changing the image format. Note: suggested workaround may impact code performance.

For example, the following migrated DPC++ code:

```
1  // migrated DPC++ code, which is using unsupported image format:
2
3  dpct::image_wrapper<cl::sycl::uint2, 2> tex; // 2-channel image is not
   supported
4
5  void test_image(dpct::image_accessor_ext<cl::sycl::uint2, 2> acc) {
6    cl::sycl::uint2 tex_data;
7    tex_data = acc.read(0, 0);
8  }
```

Refer to Compatibility Tool – Diagnostic Reference

Image format adjustment

```
texture<int, 2> textureD;
__global__ void kernel(int *dOutput,
int width)
{
  int row = threadIdx.y;
  int col = threadIdx.x;
  dOutput[row * width + col] =
tex2D(textureD, col, row);
}
```

```
/*
DPCT1059:2: SYCL only supports 4-channel
image format. Adjust the code.
*/
dpct::image_wrapper<sycl::int4, 2> textureD;

void kernel(int *dOutput, int width,
sycl::nd_item<3> item_ct1,
dpct::image_accessor_ext<sycl::int4, 2>
textureD)

{
  int row = item_ct1.get_local_id(1);
  int col = item_ct1.get_local_id(2);
  dOutput[row * width + col] =
textureD.read(col, row)[0];
}
```

Input data layout adjustment

```
int *d; size_t pitch;
cudaMallocPitch(&d, &pitch, width *
sizeof(int), height);

cudaMemcpy2D(d, pitch, h, width *
sizeof(int), width * sizeof(int),
height, cudaMemcpyHostToDevice);




cudaChannelFormatDesc channel =
cudaCreateChannelDesc<int>();

cudaBindTexture2D(NULL, &textureD,
d, &channel, width, height, pitch);
```

```
int *d; size_t pitch;
d = (int *)dpct::dpct_malloc(pitch, width *
sizeof(sycl::int4), height);

dpct::dpct_memcpy(d, pitch, h, width *
sizeof(sycl::int4), width*sizeof(sycl::int4),
height, dpct::host_to_device);
/*
DPCT1059:3: SYCL only supports 4-channel
image format. Adjust the code.
*/
dpct::image_channel channel =
dpct::image_channel::create<sycl::int4>();

textureD.attach(d, width, height, pitch,
channel);
```

Kernel invocation

```
kernel
<<<1,(width,height)>>>
(dOutput, width);
```

```
sycl::range<3> PerBlock(1, height, width);
q_ct1.submit([&](sycl::handler &cgh) {
  auto textureD_acc =
textureD.get_access(cgh);
  auto textureD_smpl =
textureD.get_sampler();
  cgh.parallel_for(
    sycl::nd_range<3>(PerBlock,PerBlock),
    [=](sycl::nd_item<3> item_ct1) {
     kernel(dOutput, width, item_ct1,
dpct::image_accessor_ext<sycl::int4,2>
(textureD_smpl, textureD_acc));

  });
});
```

Experimental features

Use experimental features of Intel® DPC++ Compatibility Tool

- --use-experimental-features=<value>

  Comma separated list of experimental features to be used in migrated code.
  By default, experimental features will not be used in migrated code.
  The values are:

  =nd_range_barrier

  - cross group synchronization during migration.

  =free-function-queries

  - allows to get `id`, `item`, `nd_item`, `group`, `sub_group` instances.

Across group synchronization not available currently in SYCL

```
#include <cooperative_groups.h>
#include <cuda_runtime.h>

namespace cg = cooperative_groups;

__global__ void kernel() {
  cg::grid_group grid = cg::this_grid();
  grid.sync();


  printf("Hello\n");
}

int main() {

  kernel<<<16, 4>>>();
  cudaDeviceSynchronize();

  return 0;
}
```

```
void kernel(const sycl::stream &stream_ct1) {
  /*
  DPCT1087:2: DPC++ currently does not support cross group
synchronization, you
  can specify "--use-experimental-
features=nd_range_barrier" to use the dpct
  helper function nd_range_barrier to migrate grid.sync().
  */
  cg::grid_group grid = cg::this_grid();
  /*
  DPCT1087:0: DPC++ currently does not support cross group
synchronization, you
  can specify "--use-experimental-
features=nd_range_barrier" to use the dpct
  helper function nd_range_barrier to migrate grid.sync().
  */
  grid.sync();
}
```

--use-experimental-features=nd_range_barrier

```
void kernel(sycl::nd_item<3> item_ct1, sycl::ONEAPI::atomic_ref<unsigned
int,sycl::ONEAPI::memory_order::seq_cst,sycl::ONEAPI::memory_scope::device,syc
l::access::address_space::global_space> &sync_ct1) {
  dpct::experimental::nd_range_barrier(item_ct1, sync_ct1);

  dpct::experimental::nd_range_barrier(item_ct1, sync_ct1);
}
dpct::get_default_queue().submit([&](sycl::handler &cgh) {
    dpct::global_memory<unsigned int, 0> d_sync_ct1(0);
    unsigned *sync_ct1 = d_sync_ct1.get_ptr(dpct::get_default_queue());
    dpct::get_default_queue().memset(sync_ct1, 0, sizeof(int)).wait();

    cgh.parallel_for(sycl::nd_range<3>(…),[=](sycl::nd_item<3> item_ct1) {
      auto atm_sync_ct1 = sycl::ONEAPI::atomic_ref<unsigned int, …seq_cst,
            …memory_scope::device,…address_space::global_space>(sync_ct1[0]);
        kernel(item_ct1, atm_sync_ct1);
      });
  });
  dpct::get_current_device().queues_wait_and_throw();
```

Functor migration – default behavior

```
struct SplitNodes
{
    int maxPointsPerNode;
    int* node_count;
    int* nodes_allocated;
    int* out_of_space;
    int* child1_;
    int* parent_;
    SplitInfo* splits;

    __device__
    void operator()( thrust::tuple<int&, int&,SplitInfo&,float4&,float4&, int> node )
    {
        int& parent=thrust::get<0>(node);
        int& child1=thrust::get<1>(node);
        SplitInfo& s=thrust::get<2>(node);
        const float4& aabbMin=thrust::get<3>(node);
        const float4& aabbMax=thrust::get<4>(node);
        int my_index = thrust::get<5>(node);
        bool split_node=false;
        // first, each thread block counts the number of nodes that it needs to alloca
        __shared__ int block_nodes_to_allocate;
        if( threadIdx.x== 0 ) block_nodes_to_allocate=0;
        __syncthreads();
```

Code snippet from - https://github.com/flann-lib/flann/blob/master/src/cpp/flann/algorithms/kdtree_cuda_builder.h

```
struct SplitNodes
{
    int maxPointsPerNode;
    int* node_count;
    int* nodes_allocated;
    int* out_of_space;
    int* child1_;
    int* parent_;
    SplitInfo* splits;

    void operator()(std::tuple<int &, int &, SplitInfo &, sycl::float4 &,
                               sycl::float4 &, int>
                        node,
                    sycl::nd_item<3> item_ct1, int *block_nodes_to_allocate,
                    int *block_left,
                    bool *enough_space) // float4: aabbMin, aabbMax
    {
        int &parent = std::get<0>(node);
        int &child1 = std::get<1>(node);
        SplitInfo &s = std::get<2>(node);
        const sycl::float4 &aabbMin = std::get<3>(node);
        const sycl::float4 &aabbMax = std::get<4>(node);
        int my_index = std::get<5>(node);
        bool split_node=false;
        // first, each thread block counts the number of nodes that it needs to

        if (item_ct1.get_local_id(2) == 0) *block_nodes_to_allocate = 0;
        item_ct1.barrier();
```

Question: how can I pass sycl::nd_item from oneDPL e.g. std::for_each(policy, iter1, iter2, SplitNodes())?

After migration with basic options of DPCT

--use-experimental-features=free-function-queries

```
struct SplitNodes
{
    int maxPointsPerNode;
    int* node_count;
    int* nodes_allocated;
    int* out_of_space;
    int* child1_;
    int* parent_;
    SplitInfo* splits;

    __device__
    void operator()( thrust::tuple<int&, int&,SplitInfo&,float4&,float4&, int> node )
    {
        int& parent=thrust::get<0>(node);
        int& child1=thrust::get<1>(node);
        SplitInfo& s=thrust::get<2>(node);
        const float4& aabbMin=thrust::get<3>(node);
        const float4& aabbMax=thrust::get<4>(node);
        int my_index = thrust::get<5>(node);
        bool split_node=false;
        // first, each thread block counts the number of nodes that it needs to alloca
        __shared__ int block_nodes_to_allocate;
        if( threadIdx.x== 0 ) block_nodes_to_allocate=0;
        __syncthreads();
```

Code snippet from - https://github.com/flann-lib/flann/blob/master/src/cpp/flann/algorithms/kdtree_cuda_builder.h

```
void operator()(std::tuple<int &, int &, SplitInfo &, sycl::float4 &,
sycl::float4 &, int> node,
int *block_nodes_to_allocate, int *block_left, bool *enough_space)
{
  auto item_ct1 = sycl::ext::oneapi::experimental::this_nd_item<3>();
  int &parent = std::get<0>(node);
  int &child1 = std::get<1>(node);
  SplitInfo &s = std::get<2>(node);
  const sycl::float4 &aabbMin = std::get<3>(node);
  const sycl::float4 &aabbMax = std::get<4>(node);
  int my_index = std::get<5>(node);
  bool split_node=false;

  if (item_ct1.get_local_id(2) == 0) *block_nodes_to_allocate = 0;
  item_ct1.barrier();
```

Workarounds

MACRO

- In some complex MACRO cases, DPCT may not be able to correctly deduce the original source line of MACRO
- Simplify complex MACROs in original CUDA code, if migration errors found related to this.

Clang differences

- nvcc-specific behaviors – This issue will be fixed in next release.
  [Rodinia] particlefilter - spaces in the triple brackets of kernel invoking e.g. "<<<` `< and >>>` `>"

Warp with mask not supported by DPCT

```
/*
 DPCT1086:0: Migration of __activemask is not supported. You may need to use 0xffffffff instead or
adjust the code.
 */
int __all_sync(unsigned mask, int predicate);
int __any_sync(unsigned mask, int predicate);
unsigned __ballot_sync(unsigned mask, int predicate);
unsigned __activemask();
```

```
214                                                    {
215     jforce.x = WARP_SHUFFLE(WARP_FULL_MASK, jforce.x, (threadIdx.x+1)&(WARPSIZE-1), WARPSIZE);
216     jforce.y = WARP_SHUFFLE(WARP_FULL_MASK, jforce.y, (threadIdx.x+1)&(WARPSIZE-1), WARPSIZE);
217     jforce.z = WARP_SHUFFLE(WARP_FULL_MASK, jforce.z, (threadIdx.x+1)&(WARPSIZE-1), WARPSIZE);
218     if (doSlow) {
219       jforceSlow.x = WARP_SHUFFLE(WARP_FULL_MASK, jforceSlow.x, (threadIdx.x+1)&(WARPSIZE-1), WARPSIZE);
220       jforceSlow.y = WARP_SHUFFLE(WARP_FULL_MASK, jforceSlow.y, (threadIdx.x+1)&(WARPSIZE-1), WARPSIZE);
221       jforceSlow.z = WARP_SHUFFLE(WARP_FULL_MASK, jforceSlow.z, (threadIdx.x+1)&(WARPSIZE-1), WARPSIZE);
222     }
```

[NAMD] WARP_SHUFFLE with WARP_FULL_MASK could be migrated by DPCT
https://www.ks.uiuc.edu/Research/namd/doxygen/CudaComputeNonbondedKernel_8cu.html

Build script in python

- Intercept-build doesn't support custom build scripts
- Modify python script to dump the commands used for CUDA code build to get a simple Makefile
- Then use intercept-build

Notices & Disclaimers

- Intel technologies may require enabled hardware, software or service activation.
- No product or component can be absolutely secure.
- Your costs and results may vary.
-


intel